Amendments to the Specification

Please replace paragraph [0019] with the following amended paragraph.

[0019] The address generator 100 includes a 32-bit adder 114 to add the segment 102 and the displacement 104. The adder 114 produces a sum output 116 and a carry bit 117 corresponding to an output carry from the 16th most significant bit position. The sum output 116 and the carry bit 117 are denoted as EARLYADD_SUM and ~~EARLYADD_COUT15~~ EARLYADD_COUT(15), respectively. Staging elements 118 and 119 are provided to buffer EARLYADD_SUM 116 and ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 117, respectively, until the base 106 and the index 108 are available. EARLYADD_SUM 116 is then added with the base 106 and the index 108 (the latter two components arriving at least one clock cycle after the segment 102 and the displacement 104) using a 32-bit 3:2 carry save adder 120. The carry save adder 120 produces a sum output 122 and a carry output 124. The sum output 122 and the carry output 124 are then added by a 32-bit adder 126 to produce the lower 16 bits of the linear address 112, as well as a carry bit 128, denoted as ~~LA_FA_COUT15~~ LA_FA_COUT(15). ~~LA_FA_COUT15~~ LA_FA_COUT(15) 128 corresponds to a carry bit generated by the addition of the lower 16-bits in the adder 126.

Please replace paragraph [0020] with the following amended paragraph.

[0020] To verify correct generation of a legacy 16-bit address, the address generator 100 also comprises a 16-bit 3:2 carry save adder 130 to add the lower 16 bits of displacement 104 (buffered by the staging element 131) with the lower 16-bits of the base 106 and the lower 16 bits of the index 108. The carry save adder 130 produces a sum output 132 and carry output 134. The sum output 132 and carry output 134 are then added by a 16-bit adder 136 to produce the effective address 110, as well as a carry bit 138, denoted as ~~EA_FA_COUT15~~ EA_FA_COUT(15).

Please replace paragraph [0021] with the following amended paragraph.

[0021] For the case of 16-bit addition, each 32-bit adder 114, 120 and 126 is able to block the output carry generated by the lower 16 bits from propagating to the upper 16 bits, thus maintaining the proper truncation and wrap-around properties of 16-bit, binary arithmetic. The blocking of the appropriate output carry bit is controlled by the input ~~ASIZE16~~ ASIZE(16) 140 into the address generator 100. The input ~~ASIZE16~~ ASIZE(16) 140 is set to a logic ONE for the case of 16-bit address generation, and to logic ZERO for the case of 32-bit address generation.

Please replace paragraph [0022] with the following amended paragraph.

[0022] However, as the segment 102 is a 32-bit binary number, the addition of it to the other address components may result in a carry that needs to propagate from the lower 16 bits to the upper 16 bits in the generation of the linear address 112. The input ~~ASIZE16~~ ASIZE(16) 140 may cause all such carries to be blocked and, therefore, the correction logic 142 is needed to determine if a carry bit equal to a logic ONE should be added to the upper 16 bits during the calculation of the linear address 112. The correction logic 142 accepts as input ~~ASIZE16~~ ASIZE(16) 140, as well as the output carry bits from the adders 114, 120, 126, 130 and 136 resulting from the lower 16 bit additions. Specifically, these inputs include: ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 117, ~~LA_FA_COUT15~~ LA_FA_COUT(15) 128, ~~EA_FA_COUT15~~ EA_FA_COUT(15) 138, the 16th most significant output carry bit of the carry save adder 120, denoted as ~~LA_CSA_COUT15~~ LA_CSA_COUT(15) 144, and the 16th most significant output carry bit of the carry save adder 130, denoted as ~~EA_CSA_COUT15~~ EA_CSA_COUT(15) 146. The correction logic 142 examines the output carry bits to determine if all carry bits correspond to the generation of the effective address 110. If so, no additional carry is needed to generate the upper 16 bits of the linear address 112, and, thus, the output 148 of the correction logic 142, denoted as ~~COUT15~~ COUT(15), is set to a logic ZERO. Otherwise, an additional carry is needed, and ~~COUT15~~ COUT(15) 148 is set to a logic ONE. The output ~~COUT15~~ COUT(15) 148 of the correction logic 142 then drives the input carry ~~CIN16~~ CIN(16) 150 for the addition of the upper 16 bits in the adder 126.

Please replace paragraph [0023] with the following amended paragraph.

[0023] As can be seen in FIG. 1, the correction logic 142 lies directly in the critical path for generation of the upper 16 bits of the linear address 112. Thus, the propagation delay of the signals through the correction logic 142 places a limitation on the rate at which 16-bit addresses may be generated. Even worse, the propagation delay through the correction logic also ~~impact~~ impacts 32-bit address generation, and, thus, overall processor speed.

Please replace paragraph [0024] with the following amended paragraph.

[0024] An example processor 200 to address some of the limitations described above is shown in FIG. 2. The processor 200 includes an instruction scheduler 202 to schedule instructions for execution. To prepare an instruction for execution, the instruction scheduler 202 passes a set of address components 204, such as a segment, a displacement, a base and an index, to ~~the~~ an address generator 206. The address generator 206 may, for example, generate one or more linear addresses for the instruction using the aforementioned address components 204. The linear address, for example, may point to a physical location in memory wherein an argument of the instruction is located.

Please replace paragraph [0036] with the following amended paragraph.

[0036] As shown in FIG. 3, the address generator 300 includes a 32-bit adder 314 to add the segment 302 and the displacement 304. The adder 314 produces a sum output 316 and a carry bit 317 corresponding to an output carry from the 16th most significant bit position. The sum output 316 and the carry bit 317 are denoted as EARLYADD_SUM and ~~EARLYADD_COUT15~~ EARLYADD_COUT(15), respectively. Staging elements 318 and 319 are provided to buffer EARLYADD_SUM 316 and ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 317, respectively, until the base 306 and the index 308 are available. EARLYADD_SUM 316 is then added with the base 306 and the index 308 (the latter two components arriving at least one clock cycle after the segment 302 and the displacement 304) using a 32-bit 3:2 carry save adder 320. The carry save adder 320 produces a sum output 322 and carry output 324. The sum output 322 and carry output 324 are then added by a 32-bit adder 326 to produce all 32 bits of the linear address 312, as well as an output carry bit 328, denoted as ~~LA_FA_COUT15~~ LA_FA_COUT(15). ~~LA_FA_COUT15~~ LA_FA_COUT(15) 328 corresponds to a carry bit generated by the addition of the lower 16-bits in the adder 326.

Please replace paragraph [0037] with the following amended paragraph.

[0037] To verify correct generation of a legacy 16-bit address, the address generator 300 also includes a 16-bit 3:2 carry save adder 330 to add the displacement 304 (buffered by the staging element 331) with the base 306 and the index 308. The carry save adder 330 produces a sum output 332 and a carry output 334. The sum output 332 and the carry output 334 are then added by a 16-bit adder 336 to produce the effective address 310, as well as a carry bit 338, denoted as ~~EA_FA_COUT15~~ EA_FA_COUT(15). ~~EA_FA_COUT15~~ EA_FA_COUT(15) 338 corresponds to the carry bit generated by the addition of the lower 16-bits in the adder 336.

Please replace paragraph [0038] with the following amended paragraph.

[0038] For the case of 16-bit addition, each 32-bit adder 314, 320 and 326 is able to block the output carry generated by the lower 16 bits from propagating to the upper 16 bits, thus maintaining the proper truncation and wrap-around properties of 16-bit, binary arithmetic. The blocking of the appropriate output carry bit is controlled by the input ~~ASIZE16~~ ASIZE(16) 340 into the address generator 300. The input ~~ASIZE16~~ ASIZE(16) 340 is set to a logic ONE for the case of 16-bit address generation, and to logic ZERO for the case of 32-bit address generation.

Please replace paragraph [0039] with the following amended paragraph.

[0039] However, as the segment 302 is a 32-bit binary number, the addition of it to the other address components may result in a carry that should propagate from the lower 16 bits to the upper 16 bits in the generation of the linear address 312. The input ~~ASIZE16~~ ASIZE(16) 340 may cause all such carries to be blocked and, therefore, the address generator 300 includes a force carry input 342 to allow the value of this carry to be set to a particular value at the start of address generation. In this way, generation of the 32-bit linear address is not delayed due to the determination of the correct value for this carry-bit. However, if the force-carry input 342 causes the carry bit to be set to an incorrect value, then the resulting address will also be incorrect, thereby requiring that the linear address be regenerated using a different value for the force-carry input 342.

Please replace paragraph [0040] with the following amended paragraph.

[0040] To determine if the generated linear address 312 based on the force carry input 342 is correct, the address generator 300 contains correction logic 344 to determine, for example, the correct value of the force carry input 342. Further detail pertinent to the correction logic 344 is provided below in conjunction with FIG. 4. In general, the correction logic 344 accepts as input the output carry bits from the adders 314, 320, 326, 330 and 336 resulting from the lower 16 bit additions. Specifically, these inputs include: ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 317, ~~LA_FA_COUT15~~ LA_FA_COUT(15) 328, ~~EA_FA_COUT15~~ EA_FA_COUT(15) 338, the 16th most significant output carry bit of the carry save adder 320, denoted as ~~LA_CSA_COUT15~~ LA_CSA_COUT(15) 346, and the 16th most significant output carry bit of the carry save adder 330, denoted as ~~EA_CSA_COUT15~~ EA_CSA_COUT(15) 348. The correction logic 344 examines the output carry bits to determine if all carry bits correspond to the generation of the effective address 310. If so, no additional carry is needed to generate the upper 16 bits of the linear address 312, and, thus, the correction indicator output 350 of the correction logic 344, denoted as NC for "need correction," is set to a logic ZERO. Otherwise, an additional carry is needed, and correction indicator 350 is set to a logic ONE. The correction indicator 350 of the correction logic 344 is output from the address generator 300 for use by the processor's recovery mechanism (as described above for the example processor 200 of FIG. 2) to determine whether the generated legacy address is incorrect and needs to be regenerated using a different value for the force carry input 342. The correction logic 344 may also provide a force carry value output 352 that is representative of the previous value of the force carry input 342 to aid the recovery mechanism in determining the appropriate new value for the force carry input 342.

Please replace paragraph [0041] with the following amended paragraph.

[0041] For the generation of normal 32-bit addresses, the ~~ASIZE16~~ ASIZE(16) input 340 is set so that the carries generated by the lower 16 bits are allowed to propagate to the upper 16 bits in the 32-bit adders 314, 320 and 326. Thus, the force-carry input 342 and the correction indicator 350 and the forced carry value output 352 may be ignored as the generated 32-bit address will be correct. Thus, the correction logic 344 needed for 16-bit address generation has no negative impact on 32-bit address generation.

Please replace paragraph [0042] with the following amended paragraph.

[0042] FIG. 4 depicts an example correction logic circuit 400 that could be used to implement the correction logic 300 of FIG. 3. The example correction logic circuit 400 accepts as input the following five output carry bits as described above: ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 402, ~~EA_CSA_COUT15~~ EA_CSA_COUT(15) 404, ~~LA_CSA_COUT15~~ LA_CSA_COUT(15) 406, ~~EA_FA_COUT15~~ EA_FA_COUT(15) 408 and ~~LA_FA_COUT15~~ LA_FA_COUT(15) 410. An exclusive-OR (XOR) operation (XOR gate 412) is performed on ~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 402, ~~EA_CSA_COUT15~~ EA_CSA_COUT(15) 404, and ~~LA_CSA_COUT15~~ LA_CSA_COUT(15) 406. An XOR operation (XOR gate 414) is then performed on the output of the XOR gate 412 and ~~EA_FA_COUT15~~ EA_FA_COUT(15) 408. Next, an XOR operation (XOR gate 416) is performed on the output of the XOR gate 414 and ~~LA_FA_COUT15~~ LA_FA_COUT(15) 410. Thus, the example circuit 400 performs an XOR operation on all five output carry bits.

Please replace paragraph [0043] with the following amended paragraph.

[0043] Returning to FIG. 3, one having ordinary skill in the art will recognize that the XOR operation performed by the example circuit 400, in essence, compares a first set of carry bits (~~EA_CSA_COUT15~~ EA_CSA_COUT(15) 348 and ~~EA_FA_COUT15~~ EA_FA_COUT(15) 338) generated during the computation of the effective address 310 to a second set of carry bits (~~EARLYADD_COUT15~~ EARLYADD_COUT(15) 317, ~~LA_CSA_COUT15~~ LA_CSA_COUT(15) 346 and ~~LA_FA_COUT15~~ LA_FA_COUT(15) 328) generated in the computation of the linear address 312. If the same number of carry bits are equal to logic ONE in both sets (corresponding to a logic ZERO at the output of XOR gate 416), then these carries must all correspond to the computation of the effective address. Therefore, a carry bit equal to logic ZERO was generated as a result of adding the effective address to the segment (in a mathematical sense). As a result, a force carry input 342 equal to a logic ZERO would cause a correct linear address to be generated. However, if the number of carry bits equal to a logic ONE is different in the two sets (corresponding to a logic ONE at the output of XOR gate 416), then a carry-bit equal to logic ONE must have been generated as a result of adding the effective address to the segment (in a mathematical sense). As a result, a force carry input 342 equal to a logic ONE would cause a correct linear address to be generated.

Please replace paragraph [0044] with the following amended paragraph.

[0044] Returning to FIG. 4, the example correction logic circuit 400 accepts the force carry as an input 418 and the address size ~~ASIZE16~~ ASIZE(16) as input 419. The example circuit 400 is designed assuming that, for legacy 16-bit address generation, the ~~ASIZE16~~ ASIZE(16) input 419 is a logic ONE and the first attempt at generating the linear address uses a default force carry input 418 equal to a logic ZERO. Then, an AND operation (AND gate 420) is performed on the inverse of the force carry input 418, the ~~ASIZE16~~ ASIZE(16) input 419 and the output of the XOR gate 416 to produce the correction indication output 422, which corresponds to the correction indication output 350 of FIG. 3. The ~~ASIZE16~~ ASIZE(16) input 419 is included in the AND operation so that a correction indication output 422 equal to a logic ONE can occur only for legacy 16-bit addresses. During the first generation attempt when the force-carry input 418 is equal to a logic ZERO, the correction indication output 422 is equal to the output of the XOR gate 416. Thus, if the output of the XOR gate 416 is equal to a logic ZERO, then the force carry input 418 was properly set to a logic ZERO and the correction indicator 422 is set to logic ZERO, thereby indicating that no correction is needed. If the output of the XOR gate 416 is equal to a logic ONE, then the force carry input 418 was incorrectly set to a logic ZERO and the correction indicator 422 is set to a logic ONE, thereby indicating that a correction is needed. If a correction is needed, on the second attempt the force value input will be equal to a logic ONE, thus forcing the correction indicator 422 to be a logic ZERO for the second attempt, thereby indicating that the generated address is now correct.

Please replace paragraph [0045] with the following amended paragraph.

[0045] One having ordinary skill in the art will note that other circuit configurations could be used to compare the output of the XOR gate 416 to the force carry input 418. For example, a coincidence operation (i.e., the inverse of the XOR operation) could be performed on the output of the XOR gate 416 and the force carry input 418 to determine if the generated carry value is equal to the force carry input. Alternatively or additionally, the ~~ASIZE16~~ ASIZE(16) input 419 could be removed from the example correction logic circuit 400 if the recovery mechanism is able to ignore a correction indication output 422 corresponding to a normal 32-bit address.